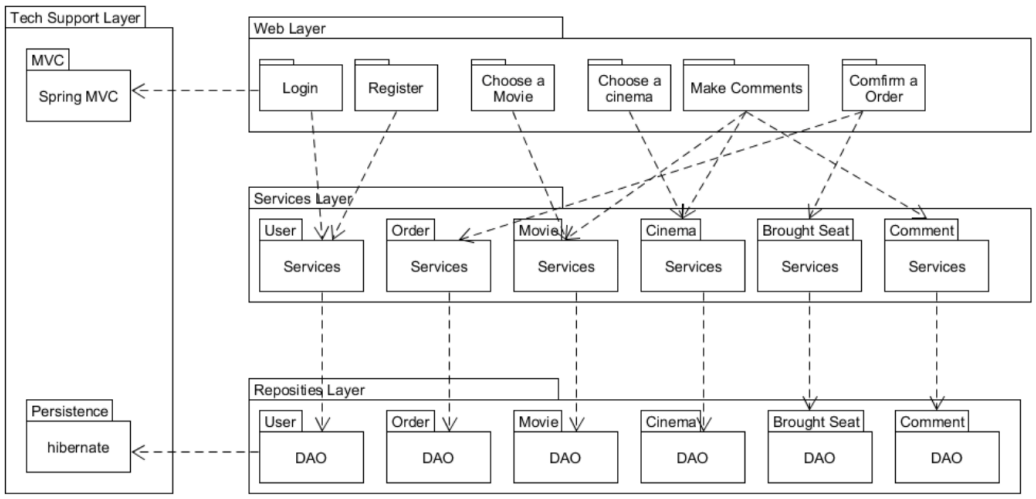
软件设计文档

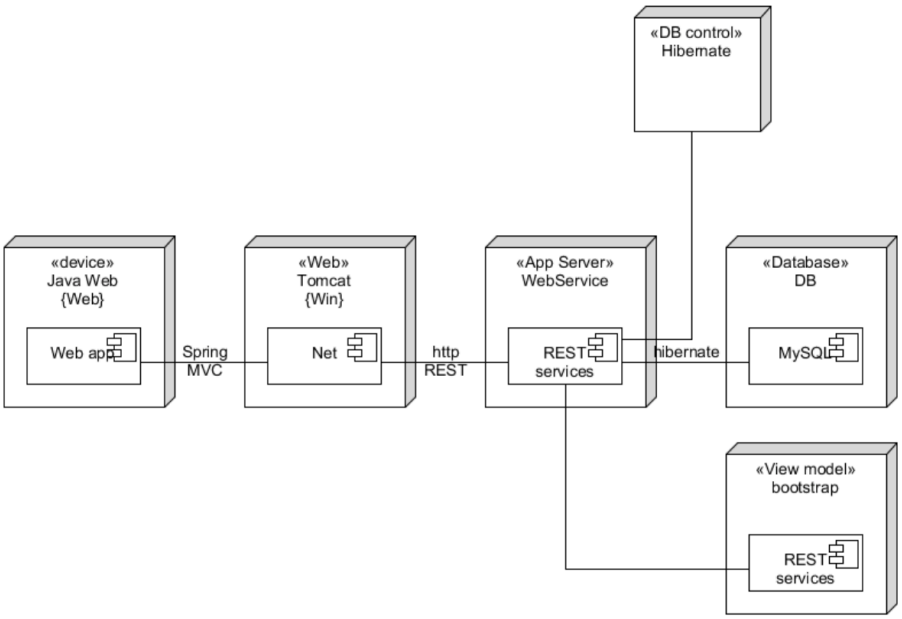
组员：陈洁璇、惠天瑞、唐文琪、高凯诗、王资、姚树航

一、总体架构设计

1、逻辑模型

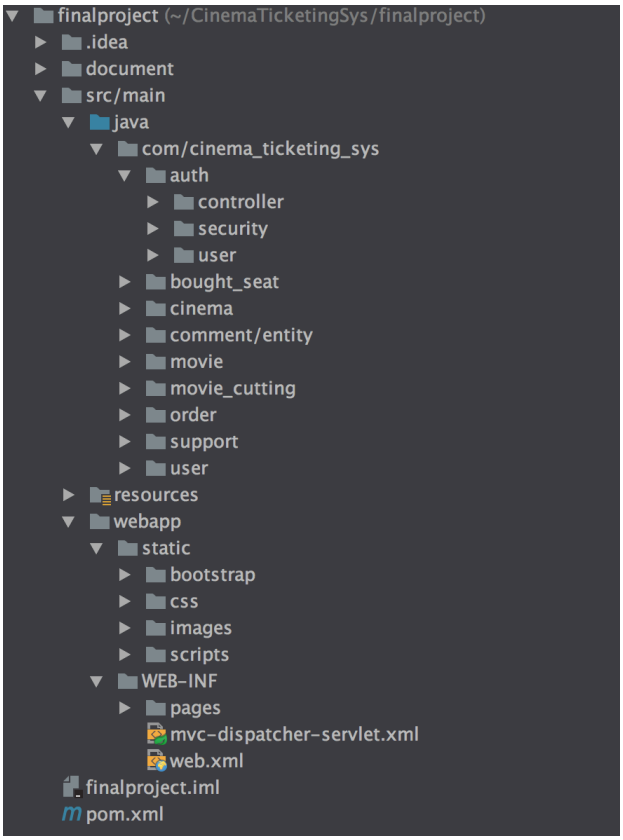


2、部署模型

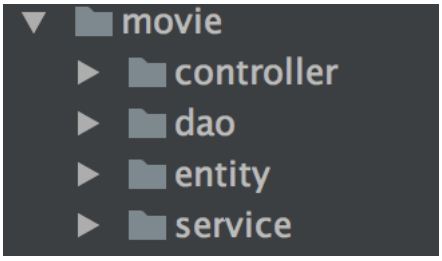


二、模块划分

整体分为前端、后端两大部分，前端部分位于 webapp 包中，后端部分位于 com/cinema_ticketing_sys 包中。如下图示，后端根据实体与其他功能划分各个模块，如实体：cinema、movie、order，验证功能 auth。



每个实体模块划分为 controller、dao、entity、service 实现。



而前端部分，static 放置 css、js 与静态资源，WEB-INF 中为页面 jsp 代码。

三、技术选型及理由

1、SpringMVC

【选型理由】

Spring 配备构建 Web 应用的全功能 MVC 框架，Spring MVC 框架用控制反转把业务对象和控制逻辑清晰地隔离，它允许以声明的方式把请求参数和业务

对象绑定。Spring 能消使配置信息一元化，实现 Restful 服务，同时支持 JDBC 和 O/R Mapping 产品（Hibernate）。

【具体实现】

位置：配置 web.xml；使用-项目后端各部分代码均有出现

SpringMVC 使用配置文件 web.xml 来引入配置，

```xml
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web
         metadata-complete="true" version="3.0">

    <display-name>Cinema Ticketing System</display-name>



    <!-- Spring的配置文件 -->
    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:/META-INF/applicationContext.xml,
            classpath:/META-INF/infrastructure.xml, classpath:/META-INF/spring-security.xml</param-v
    </context-param>

    <!-- Spring监听器 -->
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>


    <!--Spring Security过滤器-->
    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>
```

CinemaController 中的 SpringMVC

```java
@Controller
@RequestMapping("/cinemas")
public class CinemaController {

    @Autowired
    private CinemaService cinemaService;

    @RequestMapping()
    public String getAllCinemas(Integer pageNo, Integer pageSize, ModelMap modelMap) {
        List<Cinema> cinemas = new ArrayList<>();
        cinemas = cinemaService.findCinemaByPage(pageNo, pageSize);
        long totalCinemas = cinemaService.findCinemaCount(Cinema.class);

        Page<Cinema> cinemaPage = new Page<>();
        cinemaPage.setList(cinemas);
        cinemaPage.setCurrPage(pageNo);
        cinemaPage.setPageSize(pageSize);
        cinemaPage.setTotalCount((int)totalCinemas);
        cinemaPage.setTotalPage((int)Math.ceil((double) totalCinemas / pageSize));

        // ModelMap用来存储Controller处理后的数据
        // Controller将ModelMap发送到前端供JSP页面使用
        modelMap.addAttribute("cinemaPage", cinemaPage);
        return "allCinemas";
    }
```

2、Hibernate

【选型理由】

Hibernate 是一个开放源代码的对象关系映射框架，它对 JDBC 进行了非常轻

量级的对象封装，它将 POJO 与数据库表建立映射关系，可以自动生成 SQL 语句，自动执行，使得我们可以随心所欲的使用对象编程思维来操纵数据库。Hibernate 可以应用在任何使用 JDBC 的场合，既可以在 Java 的客户端程序使用，也可以在 Servlet/JSP 的 Web 应用中使用。

【具体实现】

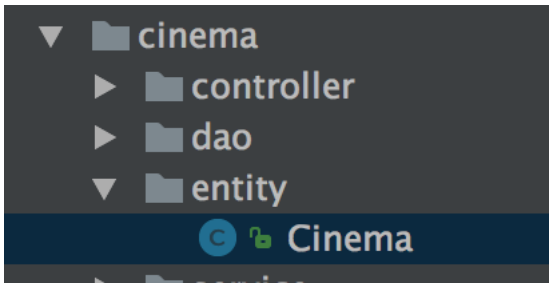
位置：配置-hibernate.cfg.xml；使用-各模块的 entity 子模块中

Hibernate 使用注解来进行映射等，例如：

@Entity　　映射实体类

@Table　　映射数句库表

在我们的项目中，Hibernate 运用到后端各个模块 entity 中，如 cinema 模块的实体




```
@Entity
@Table(name = "cinema", schema = "cinema_ticketing_db")
public class Cinema {
    private int cinemaId;
    private String cinemaName;
    private String address;
    private String description;
    private Integer rate;

    private Set<Movie> movies;

    public Cinema() {}

    // 影院 双向一对多关联 电影
    @OneToMany(cascade = CascadeType.ALL, targetEntity = Movie.class, mappedBy = "cinema")
    public Set<Movie> getMovies() { return movies; }

    public void setMovies(Set<Movie> movies) { this.movies = movies; }

    @Id
    @Column(name = "cinema_id")
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public int getCinemaId() { return cinemaId; }

    public void setCinemaId(int cinemaId) { this.cinemaId = cinemaId; }

    @Basic
    @Column(name = "cinema_name")
    public String getCinemaName() { return cinemaName; }

    public void setCinemaName(String cinemaName) { this.cinemaName = cinemaName; }
```

3、Bootstrap 框架

【选型理由】

Bootstrap 是开源的 CSS/HTML 框架，它提供了优雅的 HTML 和 CSS 规范，是由动态 CSS 语言 Less 写成。其包含了丰富的 Web 组件，如下拉菜单、按钮组、按钮下拉菜单、导航等，可以大大加快前端页面的开发。同时其网格系统也使得页面布局设计更为便捷。

【具体实现】

位置：WEB-INF/pages

bootstrap 应用到了前端各个 jsp 页面中，如每个页面的导航栏

```
<!-- 导航栏 -->
<nav class="navbar navbar-default navbar-fixed-top" role="navigation">
    <div class="container">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-targe
                    aria-expanded="false" aria-controls="navbar">
                <span class="sr-only">Toggle navigation</span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="/">小丑电影</a>
        </div>

        <div id="navbar" class="navbar-collapse collapse">

            <ul class="nav navbar-nav">
                <li class="active"><a href="/">首页</a></li>
                <li><a href="/movies?pageNo=1&pageSize=2">电影</a></li>
                <li><a href="/cinemas?pageNo=1&pageSize=2">影院</a></li>
            </ul>
```

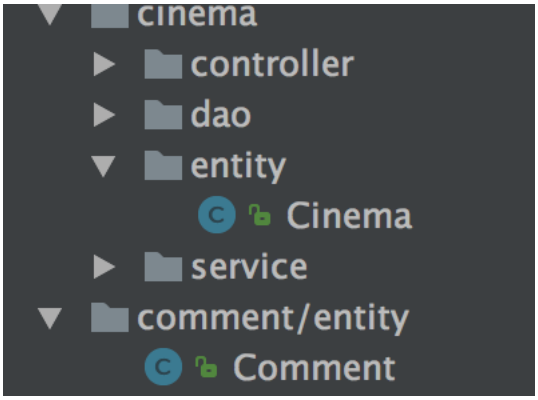
index.jsp 中网格系统的使用

```
<div class="popular-movies">
    <div class="container">
        <div class="heading">
            <h2>热门电影</h2>
        </div>

        <div class="info-movies">
            <div class="row">
                <div class="col-xs-6 col-sm-3 col-md-3">
                    <div class="movie-box box-grey">
                        <div class="inner">
                            <h5>加勒比海盗5</h5>
                            <div class="avatar">
                                <img src="/static/images/Caribbean5.jpeg">
                            </div>
                        </div>
                    </div>
                </div>

                <div class="col-xs-6 col-sm-3 col-md-3">
                    <div class="movie-box box-grey">
                        <div class="inner">
                            <h5>三只小猪</h5>
                            <div class="avatar">
                                <img src="/static/images/3pigs.jpeg">
                            </div>
                        </div>
                    </div>
                </div>
```

四、软件设计技术

1、Object-Oriented Programming 面向对象设计

本项目采用面向对象的 Java 语言，面向对象设计贯彻整个项目，将整个系统抽象为各个对象封装实现其行为功能：如影院 cinema、电影 movie 等。



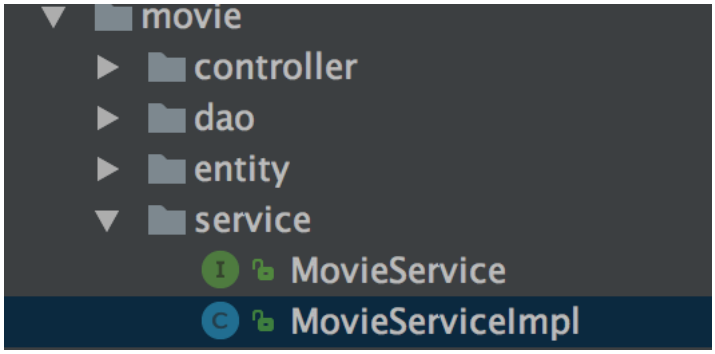
例，以下为 Movie 实现代码：

```java
@Entity
public class Movie {
    private int movieId;
    private String movieName;
    private String description;
    private Double price;
    private String posterUrl;
    private String director;
    private String cast;
    private String type;
    private String region;
    private String language;
    private Integer length;

    private Set<MovieCutting> movieCuttings = new HashSet<>();
    private Set<Comment> comments = new HashSet<>();
    private Cinema cinema;

    public Movie() {}


    // 电影 双向一对多关联 评论
    @OneToMany(cascade = CascadeType.ALL, targetEntity = Comment.class, mappedBy = "movie")
    public Set<Comment> getComments() { return comments; }

    public void setComments(Set<Comment> comments) { this.comments = comments; }

    // 电影 双向多对一关联 影院
    @ManyToOne(targetEntity = Cinema.class)
    @JoinColumn(name = "cinema_id", referencedColumnName = "cinema_id", nullable = false)
    public Cinema getCinema() { return cinema; }

    public void setCinema(Cinema cinema) { this.cinema = cinema; }

    // 电影 双向一对多关联 电影场次
    @OneToMany(cascade = CascadeType.ALL, targetEntity = MovieCutting.class, mappedBy = "movie")
    public Set<MovieCutting> getMovieCuttings() { return movieCuttings; }
```

2、Component-Oriented Programming

本项目采用了 JavaBean，是典型的组件，提高了代码复用性，其使用注解来标识 Bean。

技术使用主要在各个模块的 service 部分，例如：




```java
@Service
@Transactional
public class MovieServiceImpl implements MovieService {

    @Autowired
    private MovieDAO movieDAO;

    @Override
    public Movie findMovieById(int movieId) { return movieDAO.findMovieById(movieId); }

    @Override
    public Movie findMovieByName(String Name) { return movieDAO.findMovieByName(Name); }

    @Override
    public List<Movie> findMovieByPage(Integer pageNo, Integer pageSize) {
        return movieDAO.findMovieByPage(pageNo, pageSize);
    }
```